Programação Concorrente(5)

José C. Cunha, DI/FCT/UNL

Modelos de Servidores

José C. Cunha, DI/FCT/UNL

- Revisão do conceito de servidor
- Servidores sequenciais e concorrentes
- Conjunto fixo ou variável de processos trabalhadores

- Um Servidor é um processo que se executa concorrentemente com outros processos num SO com multiprogramação
- Cada Servidor é responsável por tratar pedidos de um determinado tipo -> um Serviço
- Exemplos:

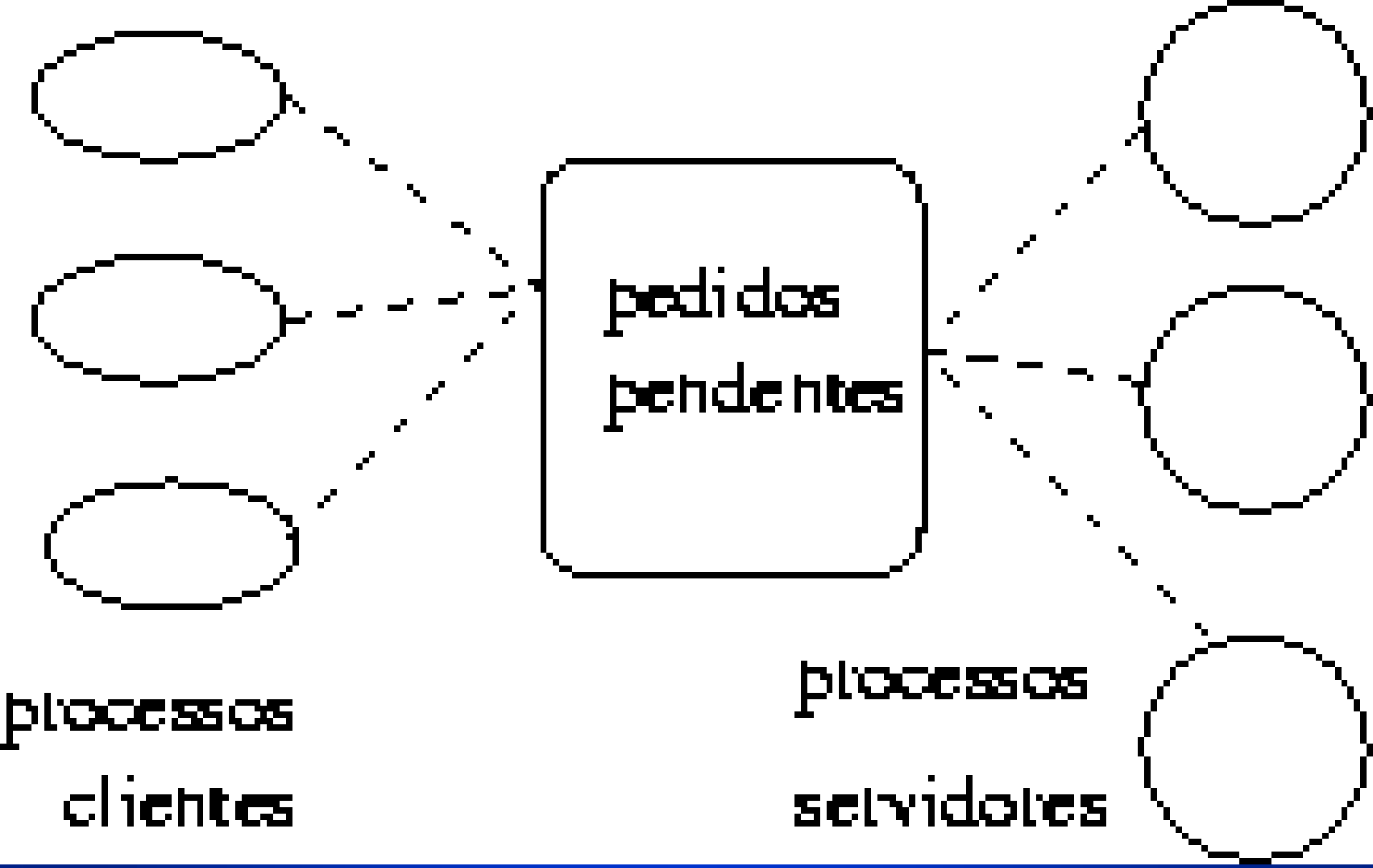
-- serviço de impressão: SPOOLer

-- serviço de ficheiros;

-- serviço de acesso a uma base de dados;

- Um mesmo Serviço pode ser tratado por múltiplos Servidores concorrentes

Que vantagens?


pedidos
pendentes
processos
clientes
processos
servidores

Clientes: -- enviam pedidos aos servidores

-- aguardam resposta (ou não)

Servidores: -- aguardam pedidos dos clientes

-- tratam cada pedido recebido

-- enviam respostas (ou não)

Operação Assíncrona:

-- os clientes vão pondo pedidos, independentes dos ritmos a que os servidores os tratam

-- os servidores podem tratar múltiplos pedidos concorrentemente

- **Conforme o tipo de pedido:**

(a) o cliente pode limitar-se a fazer o pedido e prosseguir a sua execução (por exemplo, a impressão de um ficheiro ao servidor de SPOOL)

enviar-pedido();

(b) o cliente pode ter de aguardar uma resposta (por exemplo, a consulta a um servidor de ficheiros ou a uma base de dados).

enviar-pedido(); aguardar-resposta();

Interacção clientes -- servidores

Pode ser programada por:

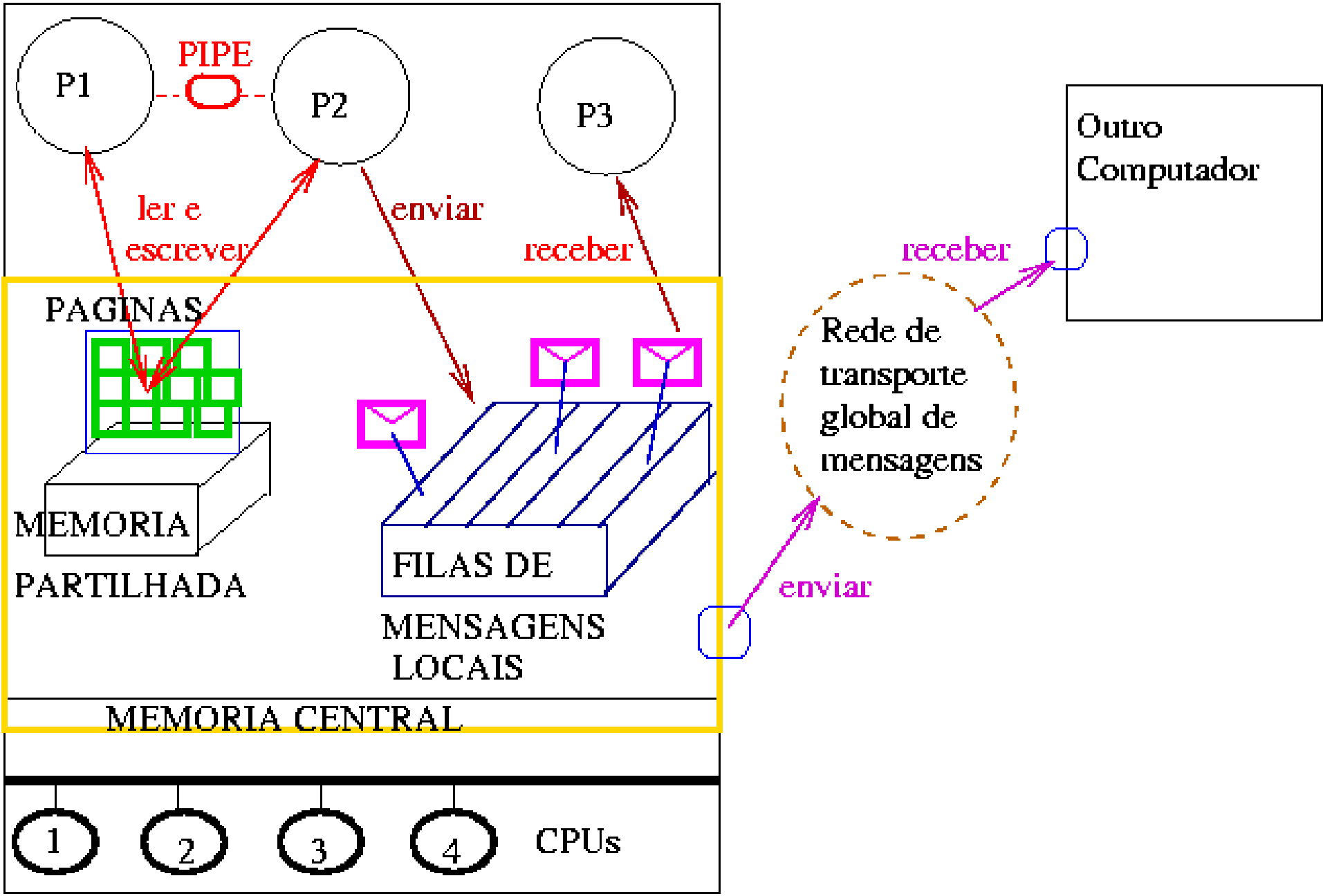
- **um modelo de comunicação baseada em memória partilhada**

  (só se o sistema for centralizado mono- ou multiprocessador)

- **um modelo de comunicação baseada em mensagens**

  (quer o sistema seja centralizado ou distribuído)

Sistema Centralizado com mono– ou multiprocessador e um BUS comum



Servidor num sistema centralizado

- **Num sistema centralizado (monoprocessador ou multiprocessador) podemos implementar a comunicação entre clientes e servidor através de uma fila de pedidos, em memória partilhada.**
- **O que é preciso programar?**

-- operações inserir/remover sobre a fila

-- exclusão mútua no acesso à fila por múltiplos processos concorrentes

-- tratar o caso em que a fila está vazia ou cheia

No caso do Unix

Fila de pedidos em memória partilhada:

-- reservar e associar-se às páginas de memória

(shmget, shmattach no Unix)

-- implementar as funções **inserir** e **remover** pedido, sobre uma fila em memória

Cliente:

```
P(exmut);
 insere-pedido();
V(exmut);
 pedido();
V(pedidos);
```

ou continua

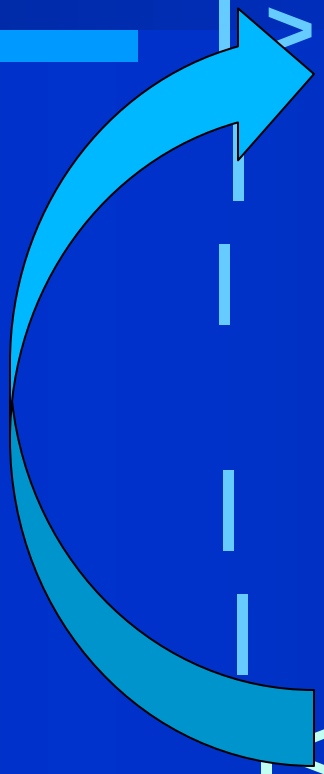
ou aguarda resposta

Servidor:

```
P(pedidos);
 P(exmut);
 remove-


 V(exmut);
 trata-pedido();
```

<- repete o ciclo;



- Uma Fila de Pedidos em Memória Partilhada
- Semáforo contador: pedidos, inicial 0
- Semáforo exclusão mútua: exmut inicial 1

Interacção Cliente-Servidor baseada em troca de mensagens

Os pedidos dos clientes e as respostas dos servidores são enviados através da invocação das operações de troca de mensagens:

enviar (mensagem,…)

receber(mensagem,…)

Mensagens:

Cliente:

```
/*pedir*/
    |
enviar(pedido );


 .....
/*se  aguarda resposta*/
receber(Resposta);
```

Servidor:

```
->  /*aguardar*/
|   receber(Pedido);
|     tratar();
|        ....
|        ....
|        ....
|   enviar(resposta);
<-     ...
```

Servidor sequencial

```
WHILE TRUE DO
  BEGIN
    receber(cliente, mensagem);
    extrair(mensagem, serviço, parametros);

     CASE serviço OF
      ......
      id: BEGIN
            executar–serviço[id](parametros,resultados);
            enviar(cliente, resultados)
          END
       ......
     ....
    END
```

Havendo múltiplos clientes concorrentes:

-- o tratamento sequencial dos pedidos pode originar apreciáveis tempos de espera,

-- a não ser que o tratamento de cada pedido seja praticamente 'imediato‘ …

por exemplo, envolva apenas a simples consulta de uma estrutura em memória

Tipos de pedidos

1- pedidos de tratamento ‘imediato’

Tipos de pedidos

1- pedidos de tratamento 'imediato'

2- pedidos que envolvem por exemplo acessos a disco, que demoram um tempo apreciável, mas determinado dentro de certos limites conhecidos (e.g. da ordem de milisegundos, no acesso a disco)

Tipos de pedidos

1- pedidos de tratamento **'imediato'**

2- pedidos que envolvem por exemplo acessos a disco, que demoram um tempo apreciável, mas **determinado dentro de certos limites conhecidos** (e.g. da ordem de milisegundos, no acesso a disco)

3- pedidos cujo tempo de tratamento é **indeterminado**

Exemplo:

- **um cliente faz um pedido a um servidor S1**
- **S1, por sua vez, precisa de pedir a um outro servidor S2 que realize uma parte do tratamento pedido**

e

- **o servidor S2 pode demorar um tempo imprevisível a responder a S1,**

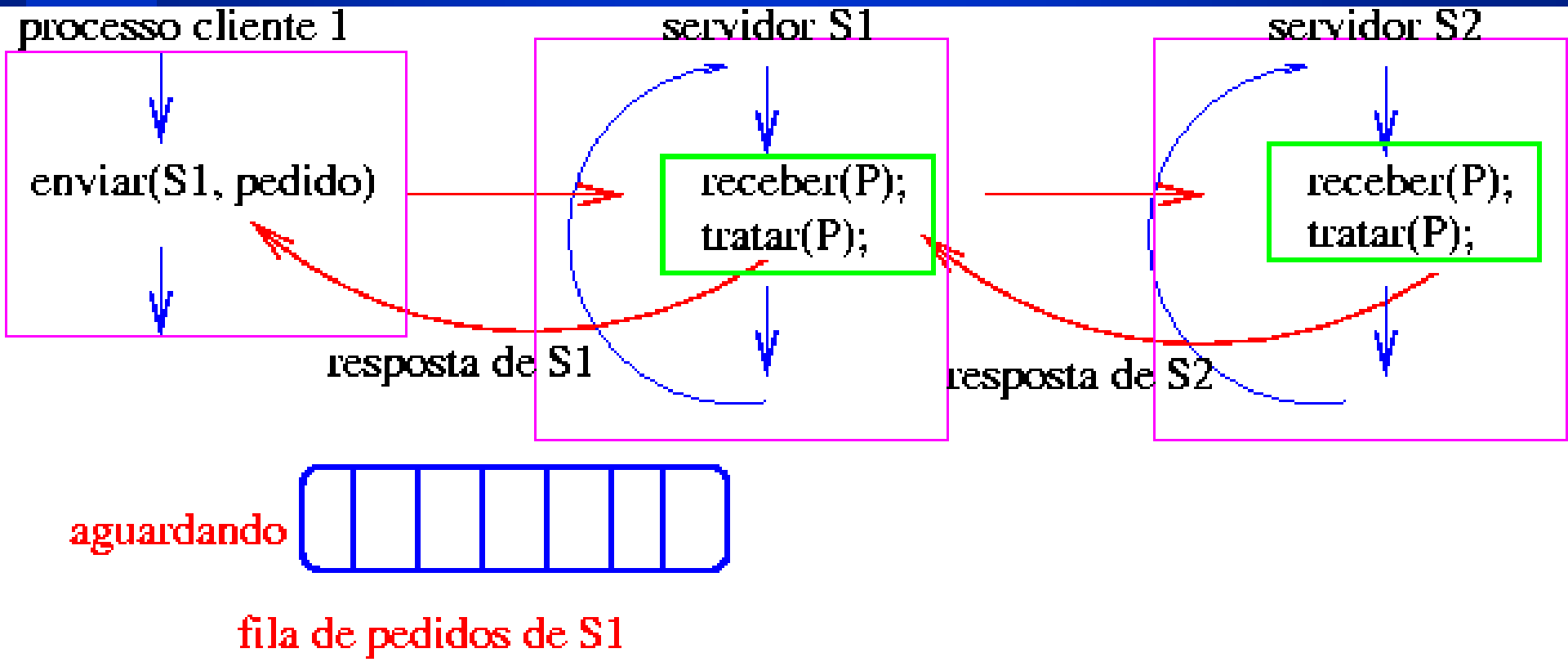
por exemplo, devido a sobrecarga de processos no sistema, num dado momento.

- Nesse caso:

-- se o servidor S1 é sequencial:

todos os pedidos de outros clientes de S1 ficam pendentes, aguardando pelo tratamento do pedido corrente, que está bloqueado aguardando a resposta de S2…

-- e os outros pedidos de S1 até poderiam ser atendidos mais rapidamente do que o corrente…


processo cliente 1
servidor S1
servidor S2
enviar(S1, pedido)
receber(P);
tratar(P);
receber(P);
tratar(P);
resposta de S1
resposta de S2
aguardando
fila de pedidos de S1

Servidor concorrente

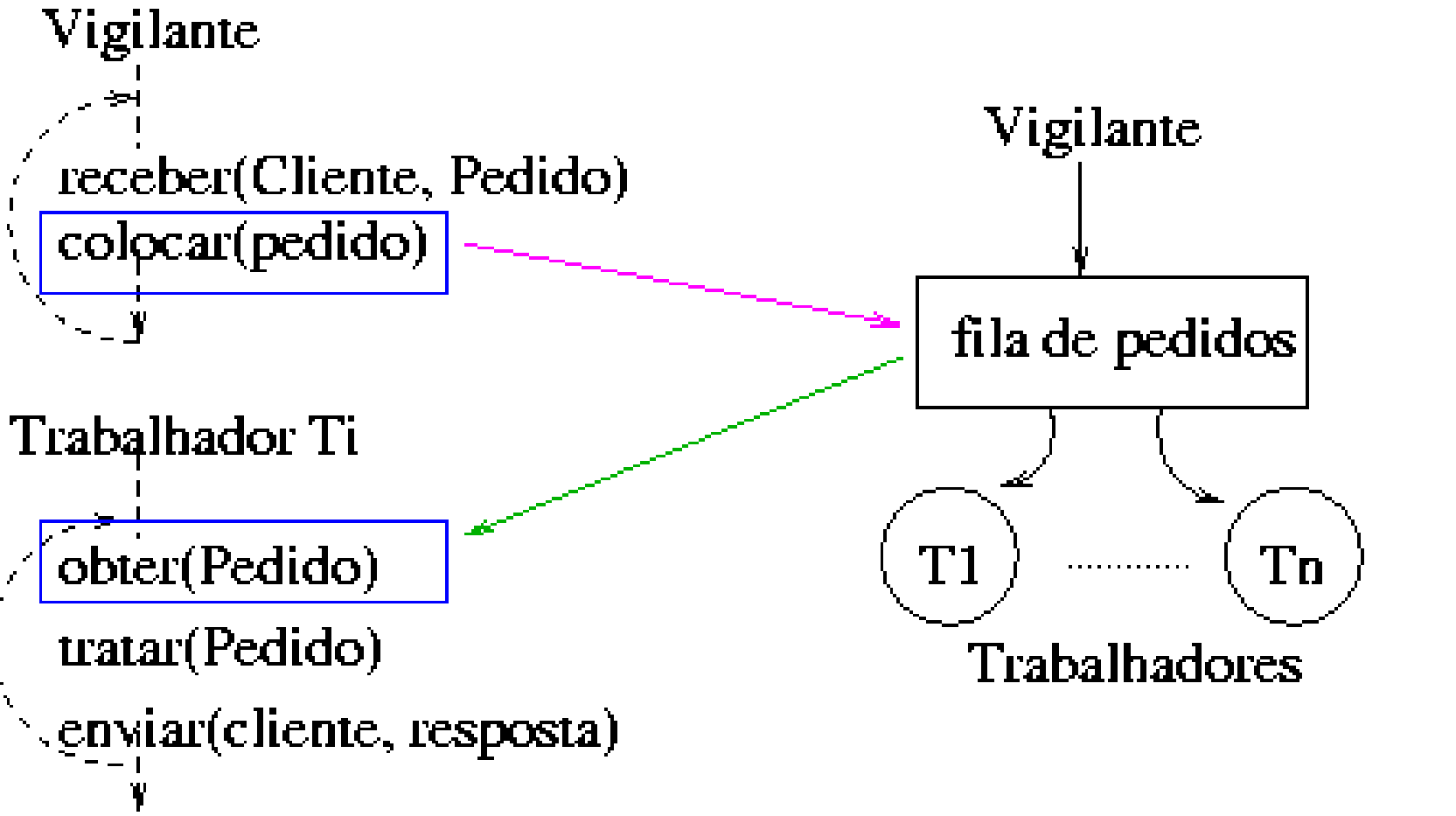
- **organizar um servidor como um conjunto de processos concorrentes, que cooperam, distribuindo entre si o tratamento dos pedidos feitos por múltiplos clientes.**

Uma possível organização:

- No início são criados N trabalhadores
- Um processo vigilante está atento à chegada de novos pedidos e encarrega-se da sua distribuição pelos processos trabalhadores disponíveis
- A distribuição dos pedidos pode ser:

-- feita directamente…

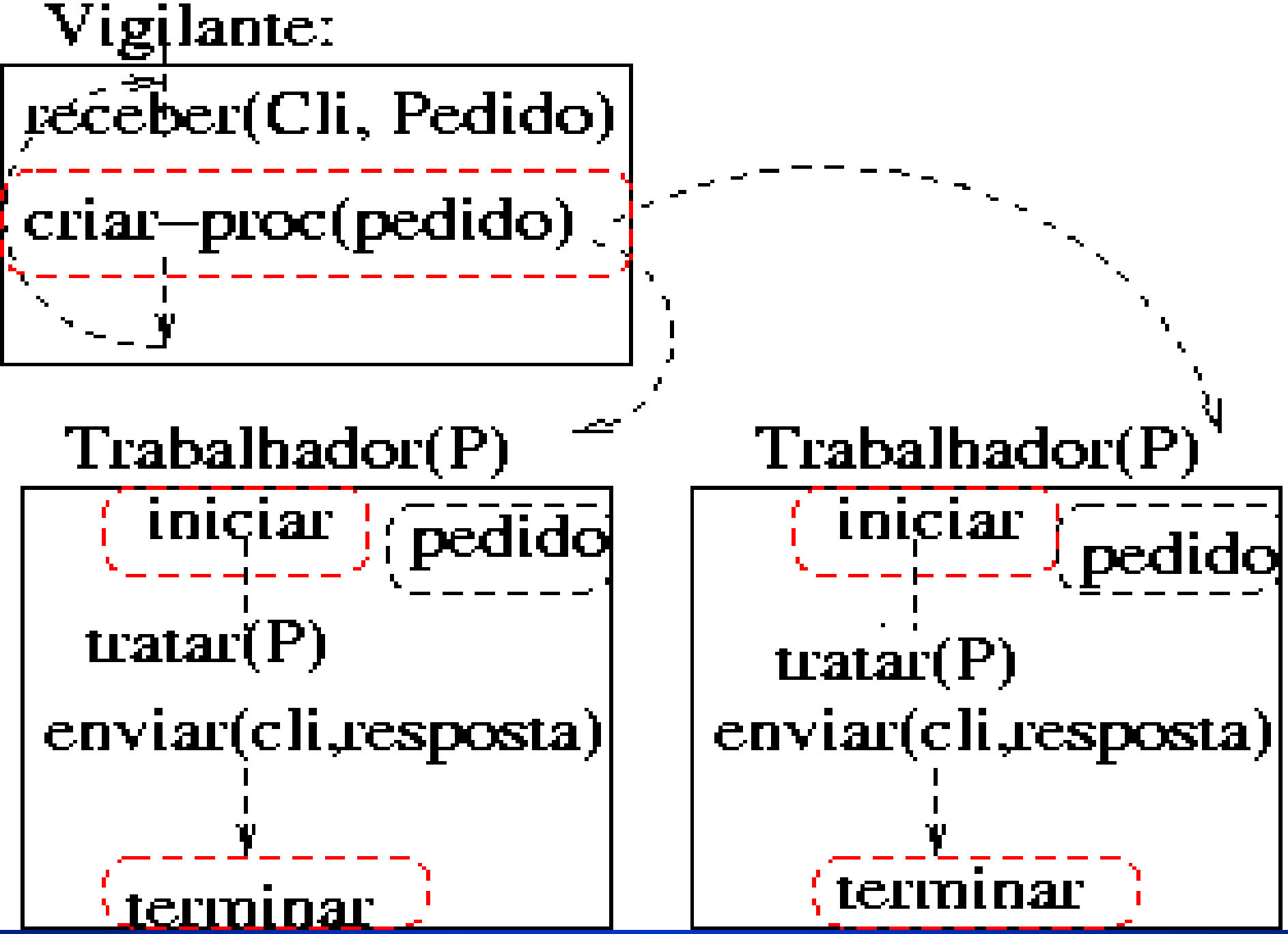
-- indirecta, via um intermediário: uma fila em memória partilhada ou uma fila de mensagens



Servidor concorrente com um conjunto fixo de trabalhadores

Outra organização possível:

- O processo vigilante está atento à chegada de novos pedidos
- Cada trabalhador pode ser criado dinamicamente, pelo processo vigilante, logo que um novo pedido chega.
- quando o trabalhador termina, o processo é destruído


Vigilante:
receber(Cli, Pedido)
criar–proc(pedido)
Trabalhador(P)
iniciar
pedido
tratar(P)
enviar(cli,resposta)
terminar
Trabalhador(P)
iniciar
pedido
tratar(P)
enviar(cli,resposta)
terminar

Outras possibilidades:

-- começar com N trabalhadores

-- conforme o número de pedidos que afluem ao servidor esteja num crescendo ou num diminuendo, assim:

--- vão-se criando novos trabalhadores ou

--- vão-se deixando terminar alguns.